

# Software Design for Particles in Incompressible Flow

Dan Martin and Phil Colella  
Applied Numerical Algorithms Group

August 8, 2003

## 1 Overview

The addition of particles to the existing Incompressible Navier-Stokes code will primarily involve the addition of the forcing function due to the particles ( $\mathcal{P}\vec{f}$ ) to the computation of the provisional velocity field  $\vec{u}^*$ . This will involve two main additions to the code:

1. The particles themselves will need to be added to the code, in the form of a `LevelData<BinFab<DragParticle>>`, where the `DragParticle` class is our application-specific derivative of the base `BinItem` class.
2. A `ParticleProjector` class will be added to compute an approximation to  $\mathcal{P}\vec{f}$  to use as a source term for the velocity update.

In addition to the `ParticleProjector` and `DragParticle` classes mentioned above, we will also define a `discreteDeltaFn` class to encapsulate the discrete  $\delta$ -function  $\delta_\epsilon$ .

We may also find it useful to define a MLC-solver class to encapsulate the MLC algorithm.

A basic diagram of the class relationships between the `Chombo` and `AMRINS`-particles classes is depicted in Figure 1.

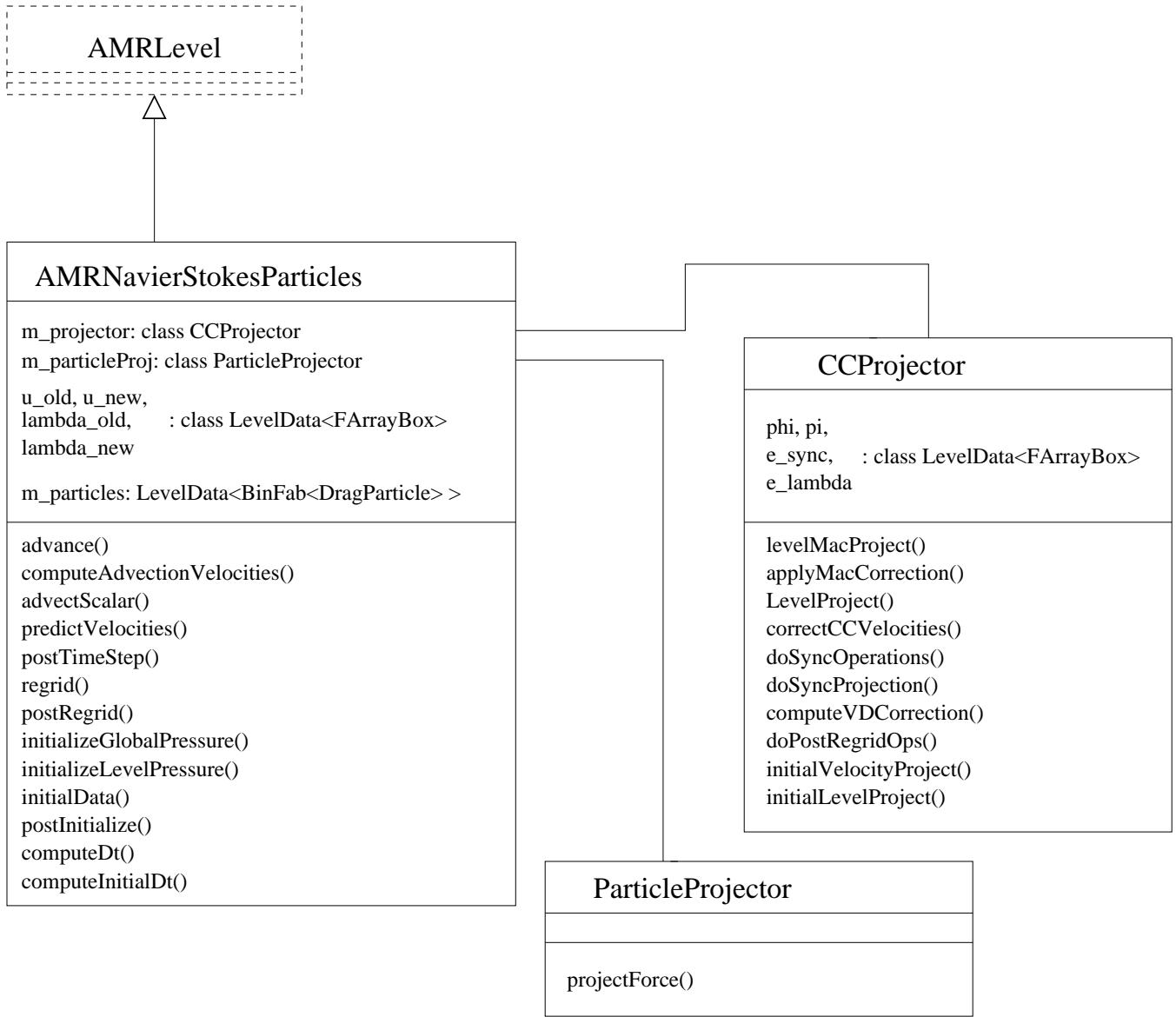


Figure 1: Software configuration diagram for the AMRINS particle code showing basic relationships between AMRINS-particle code classes and Chombo classes

## 2 Class Outline

### 2.1 The DiscreteDeltaFn class

The `DiscreteDeltaFn` class will encapsulate the discrete  $\delta$ -function used to spread the force to the mesh. This class also can compute auxiliary quantities which are functions of the numerical definition of  $\delta_\epsilon$  used.

- `Real evaluateDelta(Real a_radius)`  
Evaluate the discrete  $\delta$ -function  $\delta_\epsilon(\mathbf{r})$ .
- `Real integralDelta(Real a_radius)`  
Returns the integral of  $\delta_\epsilon$ , also known as  $Q$ :

$$Q(r) = \int_0^r \delta_\epsilon(s) s^{D-1} ds,$$

where  $D$  is `SpaceDim`.

- `Real computeK(RealVect a_radius, int a_idir, int a_jdir)`  
Returns the kernel  $K_{ij}(\mathbf{r})$ .
- `Real computeLapDelta(Real a_radius)`  
Returns  $\Delta\delta_\epsilon(r)$ . Not sure if I really need this, but it's included for completeness at the moment.

### 2.2 The DragParticle class

The `DragParticle` class will encapsulate the definition of the particles used for the application. It contains a `DiscreteDeltaFn` object to specify the spreading function. Data members include  $\vec{f}$ , the force vector for the particle, and the position and velocity of the particle,  $\mathbf{x}$  and  $\vec{u}$ .

Functions include the following:

- `void updateVel(RealVect& a_vel)`  
updates the velocity field  $\vec{u}^{(k)}$  of the particle
- `void updatePosition(RealVect& a_position)`  
updates the position  $\mathbf{x}^{(k)}$  of the particle.

- **void computeDragForce(RealVect& a\_flowVelocity)**  
computes the drag force  $\vec{f}^{(k)}$  based on the flow velocity and the particle velocity.
- **Real computeK(RealVect a\_x, int a\_idir, int a\_jdir)**  
computes  $K_{ij}^{(k)}(x)$ .
- **Real computeProjForce(RealVect a\_x, int a\_idir)**  
returns
$$\sum_{j=0}^{D-1} f_j^{(k)} K_{ij}^{(k)}(x)$$
for this particle.

### 2.3 The ParticleProjector class

The `ParticleProjector` class encapsulates the functionality needed to take the individual forces in the particles and apply them to the mesh in an approximation to  $\mathcal{P}\vec{f}$ , which may then be used as a source term for the Navier-Stokes advance.

Public Functions:

- **void define(const DisjointBoxLayout& a\_grids,  
const DisjointBoxLayout& a\_crseGrids,  
int a\_nRefCrse, Real a\_dx)**  
Defines class object.
- **void projectForce(LevelData<FArrayBox>& a\_force,  
LevelData<BinFab<DragParticle> >& a\_particles)**

Given the collection of `DragParticles` in `a_particles`, returns the projection of the force at cell centers in `a_force`, suitable for use as a source term for the INS advance.

- **void setSpreadingRadius(const a\_rad)**  
Sets spreading radius for MLC part of algorithm
- **void setCorrectionRadius(const a\_rad)**  
Sets correction radius for MLC part of algorithm.

Protected Functions:

- void computeD(LevelData<FArrayBox>& a\_D,  
                  LevelData<BinFab<DragParticle> >& a\_particles)
- void solveForProjForce(LevelData<FArrayBox>& a\_projectedForce,  
                          LevelData<FArrayBox>& a\_D)
- void computeInfiniteDomainBCs(Vector<FArrayBox>& a\_hiBCVals,  
                          Vector<FArrayBox>& a\_loBCVals,  
                          LevelData<FArrayBox>& a\_RHS,  
                          LevelData<BinFab<DragParticle> >&  
                          a\_particles)